High Performance Open Source Lagrangian Oil Spill Model

Andrea Anguiano-García[1(✉)], Olmo Zavala-Romero[1,2], Jorge Zavala-Hidalgo[1], Julio Antonio Lara-Hernández[1], and Rosario Romero-Centeno[1]

[1] Centro de Ciencias de la Atmósfera, Universidad Nacional Autónoma de México, Mexico City, Mexico
andrea_anguiano@ciencias.unam.mx, {jzavala, rosario}@atmosfera.unam.mx, julio-lara.hernandez@hdr.mq.edu.au
[2] Department of Radiation Oncology, University of Miami Miller School of Medicine, Miami, FL, USA
oszl@med.miami.edu

Abstract. An oil spill particle dispersion model implemented in Julia, a high-performance programming language, and Matlab is described. The model is based on a Lagrangian particle tracking algorithm with a second-order Runge-Kutta scheme. It uses ocean currents from the Hybrid Coordinate Ocean Model (HYCOM) and winds from the Weather Research and Forecasting Model (WRF). The model can consider multiple oil components according to their density and different types of oil decay: evaporation, burning, gathering, and exponential degradation. Furthermore, it allows simultaneous modeling of oil spills at multiple locations. The computing performance of the model is tested in both languages using an analogous implementation. A case study in the Gulf of Mexico is described.

Keywords: Julia language · Lagrangian model · Oil spill model
HYCOM · WRF · Julia vs Matlab performance

1 Introduction

The dispersion of oil spills has been of interest for many years [1]. However, after the 2010 Deepwater Horizon accident in the Gulf of Mexico, there has been more research for understanding the processes that determine the fate of oil spills. These efforts include studying, modeling and predicting oil advection, dispersion, impact of using oil dispersants, biodegradation, evaporation, controlled burning, skimming, etc. [2–5]. Other modeled processes include the rise of a buoyant plume, the spill evolution in the oceanic surface boundary layer, and the effect of waves. Currently, most of numerical models studying oil spills are based on Lagrangian modules because these numerical schemes have less diffusivity and can be run off-line with less computational effort.

The simulation of an oil spill can be separated in two stages. First, the vertical evolution of the plume from the deep ocean into ocean layers of different depth,

M. Torres et al. (Eds.): ISUM 2018, CCIS 948, pp. 118–128, 2019.
https://doi.org/10.1007/978-3-030-10448-1_11

including the surface; and second, the horizontal evolution and decay of the plume once it is in a specific layer, which depends on oil properties, presence of bacteria, application of chemical dispersants, and weathering. In the mixed layer, waves and more energetic turbulent processes can also modify the oil decay [1, 6].

The current generation of oil spill models includes the Oil Spill Contingency and Response Model [7], the Spill Impact Model Application Package/Oil Modeling Application Package [8], and the General NOAA Operational Modeling Environment [9], among others. These models have important characteristics in common. For instance, all of them have a basic structure formulated on Lagrangian methods to represent the transport processes, and they use individual algorithms for the fate processes [1].

Spaulding [1] provides a review of the current state of the art of oil spill models. In his work, the author summarizes key points learned in the design of these models: (1) the lagrangian-particle-based strategy to represent the oil allows the model to account for the temporally and spatially variation of the oil, and this strategy is amenable to sensitivity testing of results; (2) the use of tools for post-processing and visualization is essential; (3) the model should undergo validation against spill events; (4) the model needs to consider its potential use to assess the impact of response options. At the same time, the dominant driver for the transport of particles is the ocean currents and using a larger number of particles will, in general, improve the accuracy of the models. This constant search for simulating large amounts of particles, demands high computational power and efficient algorithms. Finally, the main challenges in the state-of-the-art modules are to better represent complex processes, such as the marine snow and the rising plume, as well as to improve the speed of computational algorithms while making the model user friendly.

Julia is a high-level and high-performance dynamic programming language for numerical computing that provides a sophisticated compiler, distributed parallel execution, numerical accuracy, and an extensive mathematical function library [10]. It is an open source project (https://julialang.org/) with over 500 contributors and is available under the MIT License. Julia features optional typing, multiple dispatch and good performance achieved using type inference and just-in-time (JIT) compilation, implemented using LLVM. Matlab is a commercial software that contains multiple tools for linear algebra, engineering, machine learning, plotting, etc. Matlab uses standard packages of linear algebra, and is well suited for matrix arrays and numerical computations.

High performance computing comprises not only parallel processing in clusters and GPUs, but also the development of efficient programming languages as well as the development of optimal sequential implementations of demanding algorithms. In this work, an open source Lagrangian oil spill model implemented in Julia and Matlab is described. The performance of the proposed model is analyzed through the outlined implementations.

2 Proposed Open Source Lagrangian Oil Spill Model

Software Architecture

The proposed lagrangian model was programmed in three different manners: the first code was developed in Julia; the second was implemented in Matlab using structures for storing data, and the third was also implemented in Matlab but following object oriented programming, using classes to store data.

The main stages followed by the model at each time step are:

- Read velocity fields (wind and ocean currents).
- Release particles according to the simulation setup (e.g. spill location and timing, number of particles, oil groups, oil decays).
- Interpolate velocities to the position and timing of each particle.
- Move particles (advection and turbulent-diffusion).
- Eliminate particles according to the specified oil decays (biodegradation, evaporation, burning, and gathering).

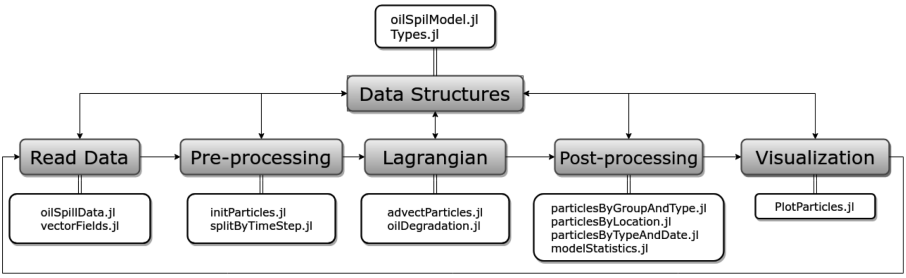

Fig. 1. Main components and associated files of the proposed oil spill model implemented in Julia.

The software architecture for all implementations follow these main steps. Figure 1 shows the modules of the Julia implementation together with the files associated with each module. A more detailed explanation of each module is available at the GitHub repository https://github.com/AndreaAnguiano/Oilspill_model-.

For both programming languages, Matlab and Julia, the same modules were created. Meticulous care was taken to make all implementations analogous. The main difference between the two Matlab implementations is the *Data Structures* module. In one case the information and status of particles is stored using structures and in the second case the information is stored in classes.

The scope of this work is to compare the performance of two programming languages, widely used in the scientific computing community, with the proposed oil spill model. The current implementations of the model run sequentially, but a parallel implementation would be valuable. In the ongoing model, the most expensive steps are reading the data and particle advection (interpolation and Runge-Kutta scheme). A straightforward parallel version will split the number of particles being advected by

the number of processing units (PU), reducing the execution time of the model and enhancing the forecasted oil spill by using more particles in the simulation. Adjustments in the *Read Data* and *Pre-processing* modules are required to achieve the suggested parallelization scheme; the other modules can be reused. It should be mentioned that a shared memory model must be guaranteed to achieve speedups of a parallel version, if a GPU is used to parallelize the evolution of the particles, sending data to the GPU (vector fields of currents and winds) can become the bottleneck of the model.

Lagrangian Scheme

A 2nd order Runge-Kutta scheme in space and time (Fig. 2) is used to compute surface and subsurface 2D displacements of oil particles due to advection and turbulent-diffusion. In this study, advection of surface oil is a function of surface ocean currents and the wind multiplied by a constant (3.5%), deflected due to Coriolis force [11]. Advection of subsurface oil is only due to subsurface ocean currents. Wind and/or ocean velocities are interpolated to the location of each particle using bi-linear and tri-linear interpolation methods. To represent sub-grid dispersion unresolved by the hydrodynamic model, turbulent-diffusion is considered by adding random velocities v' (Eq. 1) during the calculation of particle displacements.

$$v' = VR \tag{1}$$

In Eq. 1 v' is the turbulent-diffusion velocity, V is the interpolated velocity from ocean currents and wind, and R is a random number with uniform distribution from $-a$ to a, the dispersion coefficient. The value of a is defined by the user to control the degree of turbulent-diffusion [12]. Displacements of each particle are computed by multiplying the velocity of the particle (advection + turbulent-diffusion) times a time step dt defined by the user, using a 2nd order Runge Kutta scheme. For the study case in this manuscript, dt = 3600 s.

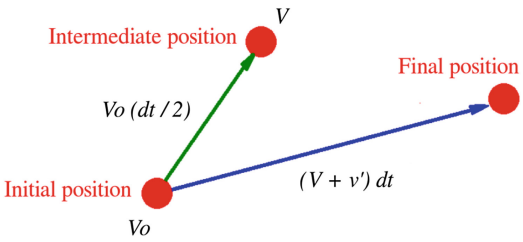

Fig. 2. 2nd order Runge-Kutta scheme. The circles represent the positions of an oil particle, ***Vo*** is the velocity interpolated to the initial position, ***V*** is the velocity interpolated to the intermediate position, ***v′*** is the turbulent-diffusion velocity, and ***dt*** is the time step.

Wind and Ocean Velocities

Previously stored wind and ocean velocities are required as inputs for the oil spill model. For the case study presented in this manuscript, winds come from simulations

carried out with the Weather Research and Forecasting (WRF) model, which is run in the supercomputing center of the Center of Atmospheric Sciences at the UNAM. The ocean currents come from the year 2010 of a 20-year reanalysis of the Gulf of Mexico obtained using the HYbrid Coordinate Ocean Model (HYCOM). This reanalysis is available for download from the HYCOM webpage (https://hycom.org/dataserver/gom-reanalysis). Additionally, wind data are spatially interpolated to the ocean model grid.

Simulation Time and Spill Locations
The simulation period is set by defining not only the initial and final day of the spill, but also the last day of the simulation because some oil remains in the ocean after the spill ends. Additionally, a probabilistic scheme for the temporal release of particles is considered to avoid rounding and therefore bias in the number of released particles. For the location of spills, several positions can be specified in terms of latitude and longitude. Particles will be randomly released around these locations, following a Gaussian distribution with a standard deviation equal to a user-defined radius in kilometers.

Oil Groups
Since a mixture of oil can be subdivided into different oil groups according to their density [13], the spill model was designed to consider as many groups as specified by the user. Each group is defined with a specific relative abundance and a specific exponential degradation rate. For instance, the first component may be assumed less dense and more abundant if a super light oil spill is simulated. Additionally, a higher exponential degradation rate could also be assigned to the first oil groups.

Oil Decay
Simulations of surface spills may include oil decay due to evaporation, burning, collection, and/or exponential degradation. Evaporation is the main natural process involved in the removal of oil from sea surface [14]. Burning and collection refers to human actions for mitigating the spill. For sub-surface simulations, only exponential degradation may be considered. The user sets the degree of exponential degradation by indicating the time in days that a fraction of oil (e.g. 95%) is expected to be degraded. This type of degradation may be used to model any degradation process following an exponential decay. The quantities of surface and sub-surface oil, as well as the amount of evaporated oil, are calculated according to the Oil Budget Calculator [2]. The removal of particles follows a probabilistic scheme to avoid rounding and therefore bias in the number of eliminated particles.

Performance Considerations
The architecture of the Lagrangian model implemented in Julia takes into consideration multiple performance tips described in the Julia language main website. The most important are: to avoid using global variables by modularizing the code in multiple functions; to reduce memory allocations by not using containers with abstract types and declaring types of each variable inside structures and functions; to pre-allocate arrays; to maximize the use of vectorized operations; and to access arrays in the proper order.

Case Study

The proposed model was used to simulate the 2010 Deepwater Horizon oil spill. The number of oil barrels spilled, the amount of oil that was evaporated, collected, and burned was obtained from the Oil Budget Calculator [2]. This simulation generates one particle for every two barrels of oil, it has a time step of one hour, and is configured to start on April 22nd and end on November 1st of 2010. It is configured to simulate 8 different types of oil at three depths (surface, 400 m and 1100 m), and the location of the spill is set at 28.73°N, 88.36°W. This simulation does not make any reinitialization of the particles positions from satellite data, as it has been proposed by Liu et al. [15]. Figure 3 shows the amount of oil at the surface integrated through time, with barrels of oil as units, in a logarithmic scale. This figure only shows the third type of oil that was simulated, with an exponential decay of 4.5 days.

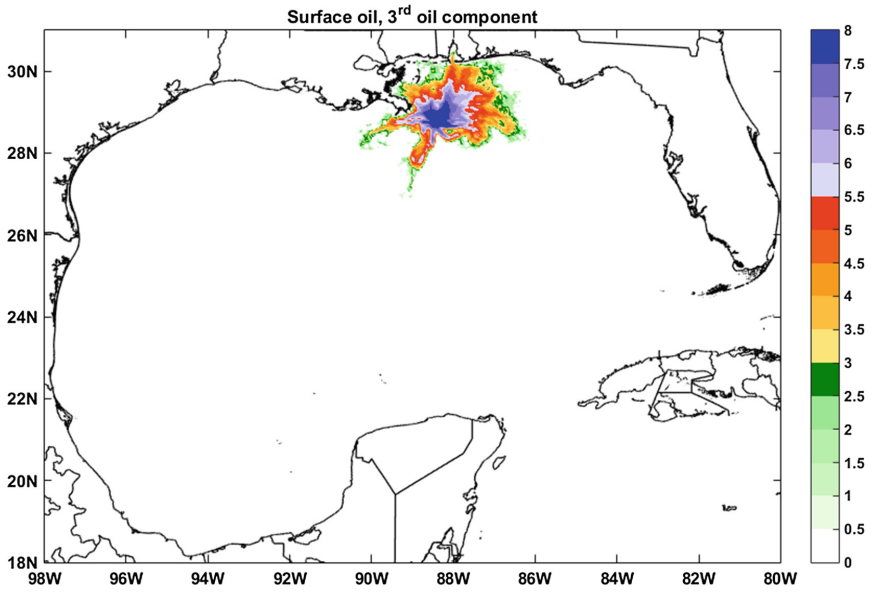

Fig. 3. Cumulative concentration of oil at the surface from a simulation of the 2010 Deepwater Horizon oil spill. The oil component displayed has an exponential decay of 4.5 days.

To assess the model performance, modeled trajectories were compared with the oil distribution mask from satellite images provided by Liu et al. [15]. The comparison shows a good agreement between the model simulation and satellite estimates. It is important to mention that the main forcing moving the oil is the ocean currents, using currents from other ocean models could produce different results. Figure 4 shows the oil spill distribution in four different dates; in the upper row, the oil masks at the surface obtained from the MODIS satellite are shown, and in the bottom row are the results from the model simulation. The different colors indicate the type of oil being simulated.

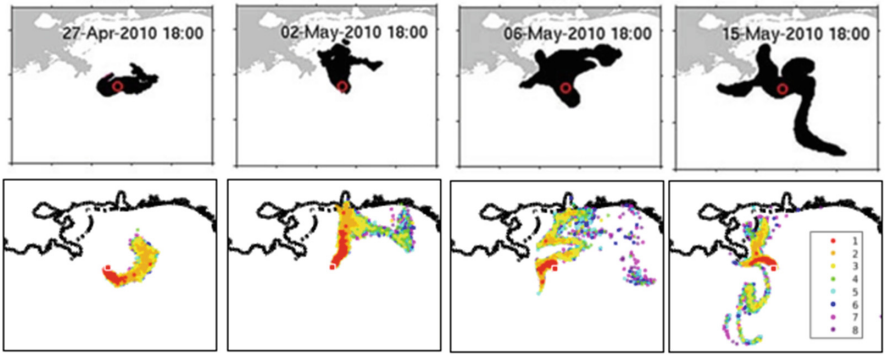

Fig. 4. Oil distribution comparison between satellite images (top) and the numerical simulation (bottom) for four different dates. Oil types are shown with different colors according to the legend. The oil masks from satellite images were adapted from Liu et al. [15]. (Color figure online)

3 Performance Analysis

The performance of the proposed model was tested using two equivalent versions implemented in Matlab. Both Matlab codes were programmed as efficient as possible using vectorized operations and indexed arrays.

Lagrangian oil spill models are very useful in two main scenarios: (1) when a new spill happens and it is critical to predict where the oil will go in the following hours or days, and (2) when oil companies or environmental agencies need scenarios showing what could happen if an oil spill occurs under specific conditions. In this last case it is more important to model longer times, on the order of weeks, under different atmospheric and oceanic conditions.

Following these two considerations, performance tests were designed as follows: (1) by changing the number of particles used to simulate each barrel of oil, and (2) by changing the number of simulation days. For each test, the three model implementations, one in Julia and two in Matlab, were executed two times. The average of these two runs is presented in this section. The performance analysis was done in a desktop computer with an Intel Core i7-6700 processor, 16 GB of DDR4 RAM, and a GeForce GTX 1080 GPU. Model parameters are summarized in Table 1.

Table 1. Model parameters considered for the performance tests.

Parameter	Value	Parameter	Value
Wind contribution	0.035	Depths	0, 100, 1000 m
Evaporation	On	Spill location	28.73° N, 88.36° W
Burn	On	Visualization	Off
Biodegradation	On	Save output	Off
Collection	On	Runge Kutta	2nd order
Number of oil components	8	Start date	04/22/2010

The performance comparison when the number of particles per barrel is modified is presented in Fig. 5. The execution time for all the implementations increases proportionally with the number of particles per barrel of oil. The total number of particles created for the first scenario (1 particle per barrel) is in the order of 5×10^5, while for the last scenario (10 particles per barrel) is in the order of 5×10^6. The Matlab implementation using classes was only tested up to 5 particles per barrel because of its long execution times (4.6 h for 5 particles) and because the amount of RAM was depleted.

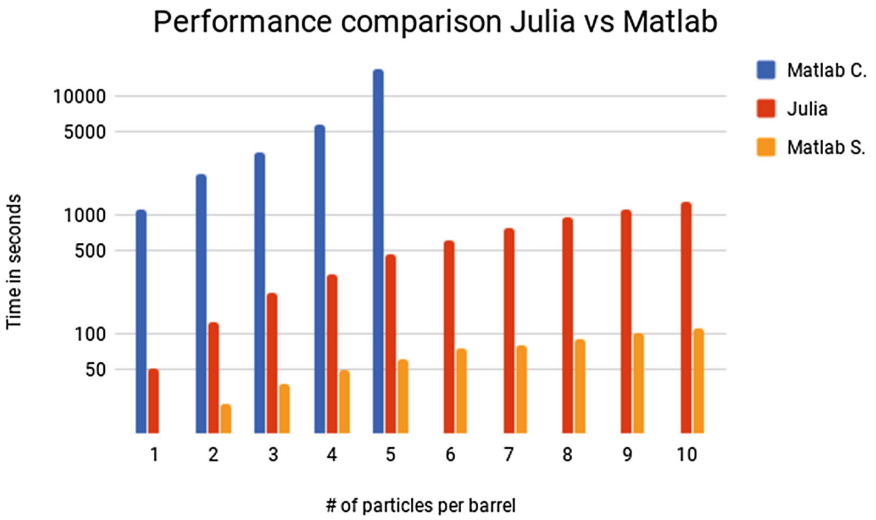

Fig. 5. Model performance comparison between three implementations: with Julia (red), Matlab using classes (blue) and Matlab using structures (yellow), by modifying the number of particles per barrel. (Color figure online)

The largest speedup obtained between the fastest Matlab implementation and Julia implementation is 11. This first test shows that different implementations of the same model can influence the performance more than the programing language used. A deeper analysis, using profiler tools of both languages, shows that the main difference between the performance of the two Matlab versions is the time needed for searching values inside arrays of data structures and searching inside arrays of objects. Searching inside structures is much faster than searching inside classes instances. In the case of the Julia implementation, the largest amount of time is used for the interpolation of currents and winds in the Runge Kutta method. To perform 2D interpolations the *interpolations* package is used.

For the second performance test, the number of days simulated varies from 2 to 20. The time for all implementations is shown in Fig. 6. For this test, the largest speedup obtained between Matlab (using data structures) and Julia is 3.5.

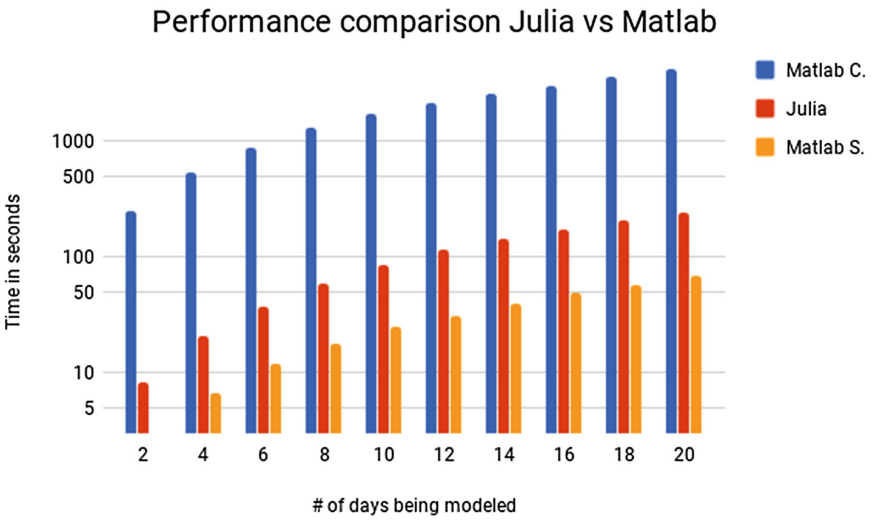

Fig. 6. Performance comparison among three implementations of the oil spill model: using Julia (red), Matlab with classes (blue), and Matlab with structures (yellow), by modifying the number of days that the oil spill is simulated. (Color figure online)

These results show that different implementations of the same model can change the performance by several orders of magnitude. They also suggest that the current Julia implementation could be optimized because of the great difference in performance compared with the optimal Matlab implementation.

One of the main aspects that influence the performance in Julia is the size of memory allocations and the time used by the garbage collector (GC) to recover unused objects. Julia provides information about these two important factors with the *@time* macro for the execution of any program, in this case the Lagrangian oil spill model. The amount of memory allocated by a program in Julia can vary greatly, depending on the implementation of the software, as described in the *performance tips* section of the documentation.

Table 2 shows the amount of memory that was allocated (in gigabytes) and the percentage of time used by the GC. These results were obtained when executing the implementation of the Lagrangian model in Julia for the first test, varying the amount of particles used to simulate each barrel of oil.

Table 2. Memory allocation and percentage of time used by the garbage collector (GC).

Particles per barrel	1	2	3	4	5	6	7	8	9	10
Percentage of GC time	49%	59%	66%	69%	73%	74%	77%	79%	79%	80%
Memory allocation (GB)	32.1	54.6	80.6	102	124	146	168	189	211	234

The large amount of memory allocations happens for two main reasons: the large amount of data read in every time step, with information of the currents and winds, and the way Julia allocates memory when copying data from one array to another variable.

4 Conclusions and Future Work

In this work, a new open source oil spill model implemented in Julia is described. The model is based on a Lagrangian algorithm for tracking particles and can consider multiple oil components and different processes of oil decay. It is initialized using oceanic currents from the HYCOM model and wind data from the WRF model, and it allows the simultaneous simulation of spills in various locations. The proposed implementation is compared with two implementations using the Matlab programming language, one based on classes and the other on structures. All the implementations are free to use and are available for download at https://github.com/AndreaAnguiano/Oilspill_model- (Julia version), https://github.com/AndreaAnguiano/OilSpill_vStruct (Matlab using structures), and https://github.com/olmozavala/oilspill (Matlab using classes).

The main conclusion of this study is that Julia, a young programming language that is being updated very fast, is not yet as mature as Matlab. In multiple occasions there is conflict between packages and the system can stay in an unstable condition. Julia contains many packages to solve the same problem, which makes it hard to decide which one is the best option. However, Julia is free, its community is growing rapidly, faster and better libraries are being created as well as improved tools for debugging and profiling, which makes it one of the best programming languages for scientific computing nowadays.

Another result that is relevant is the importance of efficient implementations, no matter the programming language been used. Comparing the best times of both programing languages, our results show that Matlab performs 10 times faster than Julia with the proposed single core implementations. Yet comparing between the two analogous Matlab implementations, their difference in performance is very large, the code that uses structures is up to 250 times faster than the one that uses classes, while Julia's performance is between these two.

In future work we plan to test the use of multiple cores in a single processor, as well as to execute the models in parallel using multiple processors in a cluster. Additionally, it is important to reduce the size of allocated memory by Julia, which requires a deeper knowledge of this programming language.

Acknowledgments. This work was funded by the CONACYT-SENER-Hidrocarburos grant 201441. This is a contribution of the Gulf of Mexico Research Consortium (CIGoM).

References

1. Spaulding, M.L.: State of the art review and future directions in oil spill modeling. Mar. Pollut. Bull. **115**, 7–19 (2017)
2. OBCS: Oil Budget Calculator: Deepwater Horizon. Books LLC (2012)

3. Liu, Z., Liu, J., Zhu, Q., Wu, W.: The weathering of oil after the Deepwater Horizon Oil spill: insights from the chemical composition of the oil from the sea surface, salt marshes and sediments. Environ. Res. Lett. **7**, 035302 (2012)
4. Beyer, J., Trannum, H.C., Bakke, T., Hodson, P.V., Collier, T.K.: Environmental effects of the Deepwater Horizon oil spill: a review. Mar. Pollut. Bull. **110**, 28–51 (2016)
5. Özgökmen, T., et al.: Over what area did the oil and gas spread during the 2010 Deepwater Horizon Oil spill? Oceanography **29**, 96–107 (2016)
6. Guo, W., Wang, Y.: A numerical oil spill model based on a hybrid method. Mar. Pollut. Bull. **58**, 726–734 (2009)
7. Reed, M., Ekrol, N., Rye, H., Turner, L.: Oil Spill Contingency and Response (OSCAR) analysis in support of environmental impact assessment offshore Namibia. Spill Sci. Technol. Bull. **5**, 29–38 (1999)
8. Spaulding, M., Kolluru, V., Anderson, E., Howlett, E.: Application of three-dimensional oil spill model (WOSM/OILMAP) to Hindcast the Braer spill. Spill Sci. Technol. Bull. **1**, 23–35 (1994)
9. Beegle-Krause, J.: General NOAA oil modeling environment (Gnome): a new spill trajectory model. In: International Oil Spill Conference Proceedings, vol. 2001, pp. 865–871 (2001)
10. Bezanson, J., Edelman, A., Karpinski, S., Shah, V.B.: Julia: a fresh approach to numerical computing. SIAM Rev. **59**, 65–98 (2017)
11. Samuels, W.B., Huang, N.E., Amstutz, D.E.: An oil spill trajectory analysis model with a variable wind deflection angle. Ocean Eng. **9**, 347–360 (1982)
12. Döös, K., Rupolo, V., Brodeau, L.: Dispersion of surface drifters and model-simulated trajectories. Ocean Model. **39**, 301–310 (2011)
13. Reddy, C.M., et al.: Composition and fate of gas and oil released to the water column during the Deepwater Horizon oil spill. Proc. Natl. Acad. Sci. **109**, 20229–20234 (2011)
14. Berry, A., Dabrowski, T., Lyons, K.: The oil spill model OILTRANS and its application to the Celtic Sea. Mar. Pollut. Bull. **64**, 2489–2501 (2012)
15. Liu, Y., Weisberg, R.H., Hu, C., Zheng, L.: Trajectory forecast as a rapid response to the Deepwater Horizon Oil spill. In: Geophysical Monograph Series, pp. 153–165. American Geophysical Union, Washington, D. C. (2011)